



Processor Abstraction in Computer System Models

Ilya Gluhovsky

Sun Microsystems,

18 Network Circle MPK18-122, Menlo Park, CA 94025

ilya.gluhovsky@sun.com

Abstract

Performance models for computer systems are widely used to evaluate architectural tradeoffs early in the design cycle and to project the behavior of a given workload on a proposed architecture. Such models typically rely on a high-level abstraction of system components and the way they interact with a given workload. A simple example is a collection of cache miss rates used to (partially) describe the memory system. In this work we propose an abstraction of processor behavior. Our goal is to discern the main attributes of overlapping cache misses with each other and with other useful work. Our basic constructs are the probability distributions of the miss interarrival times and the times between the miss and the corresponding processor stall. It will be seen that the proposed abstraction results in errors of only up to 3 percent as compared with cycle accurate simulation. Furthermore, we will show that the constructs are latency invariant. This allows one to model a variety of system interconnects which typically have many different levels of latency.

Keywords: full system simulation, memory level parallelism, processor component models, cache miss rates, queuing models.

1. Introduction

In the early stages of designing a shared memory multiprocessor it is common practice to use fast high level models to explore a broad spectrum of design options. While such models sacrifice some precision in comparison to that of cycle accurate simulation, they are particularly useful for culling large design spaces and resolving gross architectural tradeoffs [12, 9]. Simulation can then be used for detailed studies of small chosen regions.

High level models rely on a suitable abstraction of system components and the way they interact with a given workload. One of the more familiar examples is a memory system abstraction via cache miss rates corresponding to a given workload. A system model takes a set of miss rates as input and generates and routes memory system requests probabilistically based on these rates. In this paper we propose an abstraction of a processor that aggressively exploits instruction level parallelism. From the system point of view, processors differ in the cache miss processes that they generate. In what follows we assume that the L1 cache configuration is fixed and a miss refers to an L2 miss unless noted otherwise. We also assume that the L2 cache latency is fixed.

A simple in-order processor stalls as soon as a miss takes place and waits until the requested data come back. This process is accurately described by the miss rate itself. On the other hand, an out-of-order processor can continue execution after a miss is issued and can potentially issue additional misses before stalling. Therefore, the miss rates alone no longer adequately describe the miss process because the impact of a miss on the execution time depends on how long the processor can execute past a miss and on additional misses it can find while doing so. Our challenge is to describe these features in a straightforward and compact yet accurate way. The former is important for keeping a system model simple and fast so as to allow a designer to model large multiprocessors. It will also help a

designer understand the impact of the processor on the overall system performance. The latter justifies the use of the abstraction in place of a cycle accurate simulator.

Processor abstraction is used as a building block in a large system model. Large systems typically have one or more cache levels below L2 as well as a variety of interconnect options including various nonuniform memory access (NUMA) structures. In order to support design space exploration of these interconnect constructs, the processor abstraction primitives must be memory latency invariant. This way, the primitives are evaluated once using a particular (small) system or a cycle-accurate simulator in our case. The latency invariance then allows us to use these primitives to evaluate any interconnect.

We identify two important dimensions to the processor abstraction. The first dimension is the amount of time α between two consecutive misses. Smaller distances between misses make them easier to overlap, since the chance of the processor stalling between them is smaller. We stress here that we must take into account not just the mean of α , but also its variation. Chou et al. [2] observe that misses get overlapped thanks to their burstiness. If they occurred uniformly, the effectiveness of an out-of-order processor would be greatly diminished. The second dimension is the amount of time τ that the processor is able to execute past a miss. Larger τ in particular permit looking farther for overlappable miss opportunities. In this work we show that these two entities are invariant to the system latency. Furthermore, along with execution speed characterization they provide enough information for a very accurate processor abstraction.

There have been several processor abstractions proposed in the literature. In place of τ , Sorin et al. [12, 13], and Chou et al. [2] use the number of outstanding loads as a basis for the beginning of a stall period. Furthermore, Chou et al. [2] assume that an instruction miss causes an immediate stall. With such a description one-pass model run is generally insufficient. This is because once the stall ends, there can be several misses still outstanding. One then has to make a decision whether and when to stall on any of them, but the proposed abstraction does not provide information to this effect. The situation is particularly common for variable latency interconnects, for example, in the case of an L3 hit causing a short stall. Sorin et al. [12] avoid the complication by computing a weighted average of responses from multiple model runs, each with a different fixed number of outstanding loads causing a stall. This, however, results in atypical execution patterns, since no single model run pertains to the actual behavior of a processor. There are other limiting factors. Modern processors routinely use prefetching. For example, Opteron¹ processor [8] successfully prefetches most instruction references. In this case, an accurate model must necessarily contain information about the distance that the prefetch gains before the corresponding miss would stall the processor. By contrast, the model in Sorin et al. [12] stalls immediately after the last miss before the stall takes place. Similarly, Chou et al. [2] observe that the processor can typically execute past this last miss before stalling. They use parameter $Overlap_{CM}$ to denote the proportion of useful overlapped activity. The model in Sorin et al. [12] cannot take this into account. Our model corrects for this phenomenon naturally using times τ . Additionally, the model in Sorin et al. [12] assumes a memory consistency model that precludes stalls on store misses. Many current processors (Opteron, SPARC) exhibit such behavior to a nontrivial extent [3]. Modeling this in terms of Sorin et al. [12] would require maintaining and interpreting not just the number of outstanding loads, but also that of stores and the interaction of the two (not accounting for the instruction prefetch phenomenon described

¹ Henceforth, Opteron refers to AMD Opteron Processor™

above). In addition, stores typically allow the processor to execute longer before causing a stall, adding to potential model inaccuracy.

Chou et al. [2] propose an approach where an MLP (memory-level parallelism) parameter is introduced. It is defined as the mean number of overlapped misses and is obtained by making an *epoch* assumption that all misses are issued and return at approximately the same time. Sorin et al. [12] also make this assumption for their parameter estimation. Chou et al. [2] do not use their abstraction in the context of system models. However, it is attractive to consider such a possibility due to its compactness and ease of interpretation. After all, an increased amount of miss overlap is arguably the most important benefit of having an aggressive processor. This idea is used to define “blocking factors” in Sharapov et al. [11]. In our view, such an abstraction suffers from two drawbacks in addition to those discussed above. First, the epoch assumption is often inaccurate. In particular, the average distance between the first load and a later overlapped load is 27% of a typical memory latency (87 ns) used for simulation in the case of the TPCC benchmark running on an Opteron processor and is 19% in the case of SPECJBB. Second, the MLP parameter is not interconnect invariant. For example, if the workload typically allows for two overlapped misses, $MLP = 2$ in the case of a fixed latency interconnect. If, say, an L3 cache is added whose latency is 30% of the memory latency, for two misses one of which is an L3 hit and the other an L3 miss, $MLP = 1.3$ assuming the execution only resumes after the second miss. Since the execution may resume earlier, the relationship is more complicated and additional characterization is required.

Tsuei and Yamamoto proposed a high-level simulation model [14]. It is achieved by stripping down the processor to a few essential components and tracking their state. Their model is specific to a particular processor design and requires prior knowledge of the significance of processor components in their performance impact. By contrast, we elicit a compact description of the essential processor-workload interaction and do not specifically model any part of the processor architecture. Also, Tsuei and Yamamoto [14] do not address the burstiness phenomenon of load miss traffic, which is recognized as a first order effect on performance [12, 2, 4].

Karhanis and Smith [7] propose an analytic processor model. They use a fitted function that relates the number of instructions issued per cycle to the number of instructions in the instruction window to propose formulae for branch misprediction and miss penalties. This model is not suitable for use in conjunction with a system interconnect model because it assumes a constant memory latency. If one were to allow for different interconnect latencies, the overlap miss penalty formula would become invalid as, for example, the distance between misses (our distribution α) would become significant. Additionally, the paper makes a number of assumptions to make their model feasible. First, overlaps among different type misses (e.g. a load miss and an instruction fetch miss) are assumed negligible. Second, load and store misses are modeled the same way because their assumptions imply that they behave the same and that a constant time elapses between a miss and a stall. In this paper this time is summarized by distribution τ , which is shown to be quite variable. While Figures 1d-f plot distributions τ in terms of memory references, the plots in terms of time are qualitatively very similar. We also show that the loads and stores behave quite differently. The paper also studies several interconnect invariant properties, such as branch misprediction penalty, that can be accounted via one pass of a cycle-accurate simulation as proposed here.

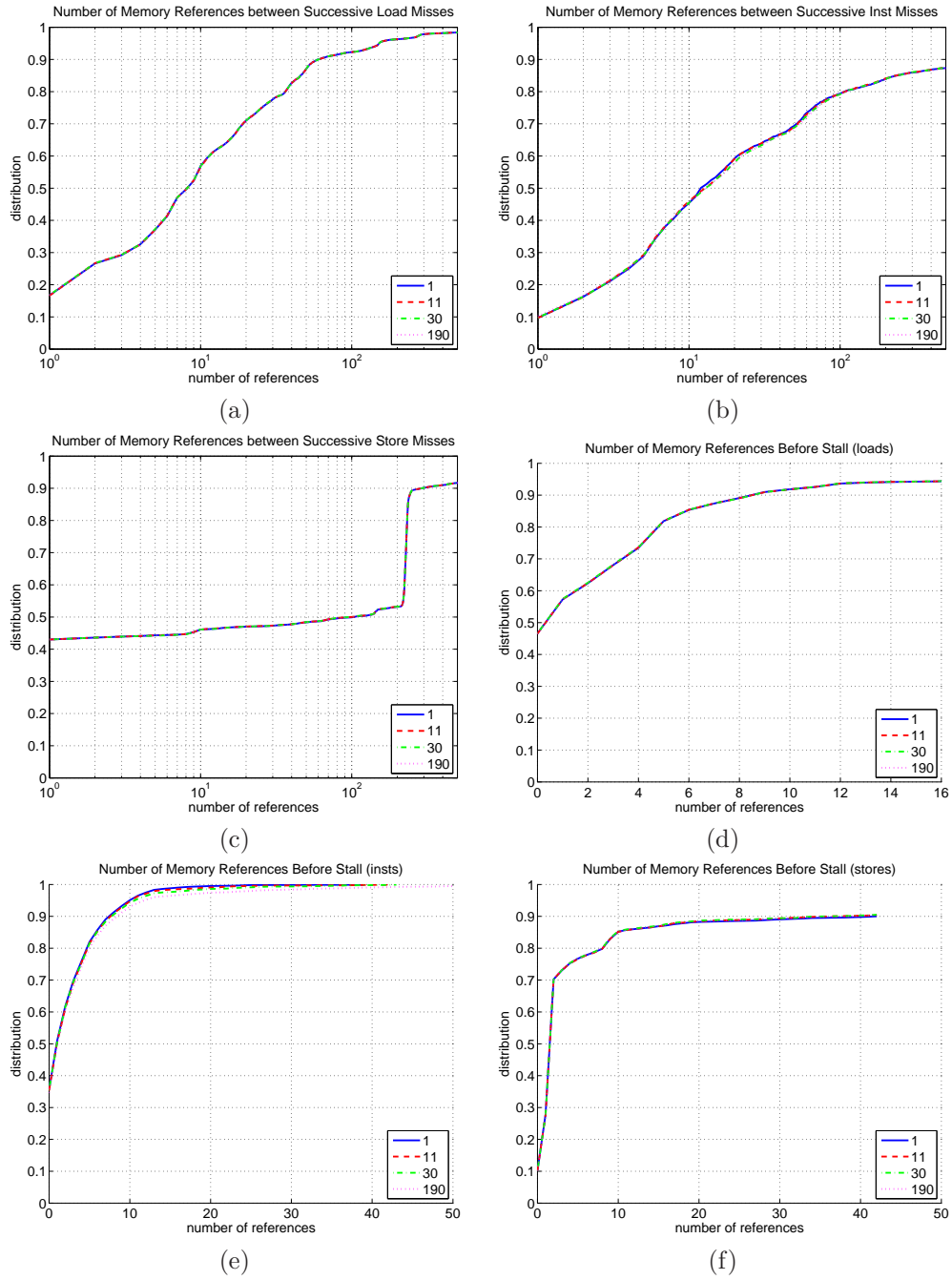


Fig. 1: TPCC abstraction distributions for 4 levels of memory latency. First row: a. α_{ld} , b. α_{if} ; Second row: c. α_{st} , d. τ_{ld} ; Third row: e. τ_{if} , f. τ_{st}

The abstractions that are discussed in this paper take a high level view of the processor with the goal of high level system modeling. In particular, these models are not meant for processor architecture design. A discussion of detailed processor modeling can be found in Barre et al. [1]. Other high level processor models and a discussion of their merits can be found in [12, 13, 7]. In particular, the latter article cites many proposals that do not allow for modeling general system interconnects.

2. Model Details

The proposed processor abstraction consists of two probability distributions for each of the three reference types (load, instruction fetch, and store). The first triplet of distributions, α_{ld} , α_{if} , and α_{st} , summarizes distances between two consecutive L2 cache misses of each type respectively. We express distance in units of instructions or memory references (hits or misses) rather than wall clock time. This is convenient because the wall clock time also includes processor stall time, which is latency dependent. Therefore, the wall clock time would have to be corrected for any constituent stall time. By contrast, no instructions or memory references are issued during stall periods obviating the need for any correction.

Note that the α distributions are still interconnect dependent because changes to the L2 cache miss rates (for example, via sharing or the number of threads in the system) would obviously change these distributions. We use them to characterize bursty miss behavior. A similar idea was employed in Sorin et al. [12] except their distributions were parameterized by the means and coefficients of variation. While we agree in principle that this should serve as a reasonable approximation in most cases, we choose not to make such a simplification in this work. The actual distributions α' used to generate load miss traffic for a particular interconnect are obtained either by rescaling the abstraction distributions α as detailed below to match the total load miss rate of the new interconnect or via cache simulation.

The second triplet of distributions τ_{ld} , τ_{if} , and τ_{st} summarizes the distance between a miss of each type and the beginning of a stall period caused by that miss respectively. We follow Sorin et al. [12] in defining a processor stall as the state in which the functional units are completely idle, and no further instructions can be retired or issued until the miss returns. We expect τ_{ld} to summarize the distance between the point when a load is issued and the point where the result is required to avoid stalling. A demand instruction miss stalls the processor immediately. Opteron processor prefetches instructions. In the case of Opteron, τ_{if} summarizes the distance between the prefetch and the would-be demand miss. In particular, this shows how the proposed framework handles prefetches. Typically, a store miss stalls a processor only if it subsequently causes the store buffer to become full. τ_{st} then summarizes the distance between a store miss and the time when the store buffer fills up. This part of the abstraction may be slightly sensitive to the interconnect detail depending on the specifics of store handling and a memory consistency model used. In the case of Opteron we show that this is largely not the case. Moreover, abstracting store behavior is usually easy and has a marginal effect on performance [3]. Sorin et al. [12] do not consider stalls on stores in their model.

The above distributions provide the core for understanding cache miss behavior that the processor generates. Since the distances are measured in the number of instructions or memory references while memory latencies and system response are measured in processor clock cycles, we must also characterize the rate of execution s_{inf} (in cycles per instruction (CPI) or cycles per reference) when the processor is not stalled. This is a standard piece

of most system models, which typically use infinite cache CPI (usually L2 cache, which includes L2 hit stalls, as in [12]).

The processor abstraction components are obtained from a single pass through a memory reference trace. The trace summarizes a cycle accurate simulation run of a processor endowed with two levels of cache and the main memory. Trace records correspond to every memory reference and include its type (load, instruction fetch, or store), the cache/memory level that supplied the data, the start and finish times and the duration of a stall period if any. As it is memory references that are captured in the trace as opposed to other instructions, we choose to measure distances in units of memory references. Alternatively, the trace could maintain an instruction count for each reference. Recording a sample from the α distributions is straightforward and amounts to recording the number of memory references from the trace between successive L2 misses of the corresponding type. For example, assuming that references are sorted according to their start times, if references 1,005 and 1,017 are successive load misses, we add 1,017-1,005=12 to the α_{ld} sample. Similarly, a τ sample is obtained by recording the number of memory references that fall between the start of a miss and the beginning of a stall period it causes.

Let us now describe how we emulate the execution for a modeled interconnect given abstraction pieces α' , τ , and s_{inf} . Assume that we know the cache miss ratios (per reference) and various latencies which describe the interconnect. Of course, the latencies are measured in time units (say, processor cycles). In general, these are sums of hardware latencies and queuing delays. We maintain both the memory reference time t_r and a wall clock time t_w as shown in Figure 2. L2 cache misses are generated using the interarrival distributions α' . If the last L2 load miss took place at time (t_r^0, t_w^0) , the next L2 load miss would occur at time $t_r = t_r^0 + m$ (see Figure 2), where m is obtained by sampling from distribution α'_{ld} , $m \sim \alpha'_{ld}$. Instruction fetch and store misses are generated analogously using α'_{if} and α'_{st} respectively. Thus, the miss processes corresponding to different reference types are statistically independent in time t_r , and evolve in the same time frame (t_r, t_w) .

Based on the interconnect miss ratios, each L2 miss is then probabilistically chosen to hit at a particular interconnect component (e.g. L3 hit, memory hit, remote L2 hit, etc.), which determines its latency l . A load miss issued at time (t_r^0, t_w^0) causes a stall at $t_r = t_r^0 + k$ (see Figure 2), that is, k references after being issued *if* it is still outstanding by then, where k is drawn from τ_{ld} , $k \sim \tau_{ld}$. To determine whether this stall takes place, we first calculate its wall clock start time t_w . Since the rate of issuing memory references is s_{inf} and k of them are issued since the miss, $t_w = t_w^0 + k \times s_{\text{inf}}$. In order for the stall to take place, t_w must be less than $t_w^0 + l$, since the latter time is when the miss comes back. This is illustrated in Figure 2.

Observe that if the next L2 load miss takes place at $t_r = t_r^0 + m$ and $m < k$ (as in Figure 2), some of the service of the latter miss is overlapped with that of the former, while if $m > k$, the latter miss is issued after the first one comes back. To determine the correct course of action in the case of $m = k$, note that a stall period corresponding to k references in the τ_{ld} sample really starts between the k th and the $(k + 1)$ th reference after the original miss. That is, the k th reference occurs before the stall begins. Therefore, when $m = k$, the miss is issued so as to precede the stall and the stall time is overlapped with servicing this later miss. At the end of a stall t_r is still $t_r^0 + k$ since no references are issued during the stall period. Instruction and store misses are modeled analogously. During a stall period, all outstanding misses irrespective of their type are serviced concurrently. In particular, the

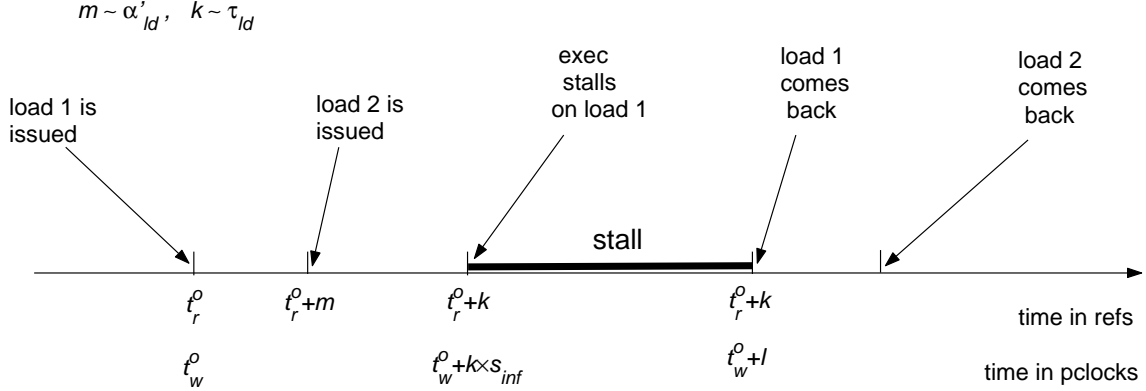


Fig. 2: Example of execution driven by the proposed abstraction. Only load activity is shown.

miss processes corresponding to different reference types are now *dependent* when observed in time t_w .

It is now clear that larger interconnect latencies impact model estimates by increasing the duration of stall periods. Larger L2 cache miss ratios pack more misses in front of a stall potentially allowing for greater miss overlap albeit more frequent stalls.

We now describe how to obtain distributions α' from abstraction distributions α . Observe that the mean of α is a reciprocal of the corresponding known miss ratio. For a given new interconnect, we use rescaled versions of α , so that their implied miss ratios match those of the modeled interconnect. Again taking loads for definiteness, this can be achieved in a straightforward manner by applying a stretching factor of m_{L2ld}/m'_{L2ld} to α_{ld} , where m'_{L2ld} and m_{L2ld} are the total L2 load miss ratios for the new and traced interconnect respectively. We found that exponentially rescaled distributions work a little better (although the difference is small). It also appeals intuitively as the logarithmic scale for miss ratios and interarrival distances is often considered natural [5]. To this end, given sample x_1, \dots, x_n from α_{ld} , we numerically find exponent ρ , such that $\sum_{i=1}^n [(1+x_i)^\rho - 1]/n = 1/m'_{L2ld}$, and use $(1+x_1)^\rho - 1, \dots, (1+x_n)^\rho - 1$ as the new sample. Alternatively, distributions α' can be obtained via cache simulation for all interconnects of interest. Note that cache miss ratios for these interconnects have to be obtained anyway. Gluhovsky and O’Krafka underscore the need for extrapolating miss rates in various circumstances to explore system configurations with large caches and/or processor counts [5]. As we have not had any experience with extrapolating these distributions, we do not advocate this method here. However, this is an interesting item for future research.

Last, let us look more closely to the way we estimate the τ distributions. It is easy to observe a sample for τ as discussed above. Note that there are some references that do not cause stalls in the trace. Such behavior is expected for many stores and references that coalesce to the same cache line with other stalling references. However, there are a number of scenarios where a reference does not cause a stall in the trace, but would cause a stall if we could observe execution long enough after it had been issued without interruption from other references. For example, suppose that load miss one is followed closely by load miss two and then causes a stall as in Figure 2. Since a considerable part of servicing the second miss is overlapped with that of the first miss, the former may not cause a stall, as it comes back soon after resumption of the execution (Figure 2). However, if miss one had

Table 1: TPCC: Relative errors of the miss frequencies for the three reference types. (a) Abstractions are for the particular latency in each case. (b) The same abstraction is used throughout.

latency	load	i-fetch	store	s_{inf}
1	1.03	1.03	1.03	7.20
11	1.03	1.03	1.03	7.25
30	1.02	1.02	1.02	7.35
190	1.00	0.99	1.00	7.25

(a)

latency	load	i-fetch	store
1	1.03	1.01	1.04
11	1.03	1.01	1.04
30	1.01	1.00	1.02

(b)

been a hit, miss two would have caused a stall. The conclusion we draw is that we observe τ samples that are biased downwards. Indeed, had miss two caused a stall before miss one did, it would have been observed instead. Thus, in that situation, we observe a quicker stall and not a slower one. We correct for this bias using the Kaplan-Meier technique [6] for censored data. For each reference we record the τ time if it is observed. If it is not observed, we record the number of references issued while the miss was outstanding and annotate it to signify that the τ time is at least as large as the recorded number. These data provide standard input to the Kaplan-Meier method.

3. Opteron Abstraction for TPCC and SPECJBB

In this section we present and evaluate the abstraction of the 2.8 GHz Opteron processor running TPCC and SPECJBB workloads. First, we show that the abstraction remains latency invariant in both cases. We carry out cycle-accurate simulation of the Opteron processor endowed with a 1MB L2 cache and the main memory whose latency is varied. Four DRAM latency levels are considered: 1 ns, 11 ns, 30 ns, and 190 ns. The corresponding average load-to-use system latencies are 58 ns, 68 ns, 87 ns, and 247 ns respectively. The columns in the middle of Tables 1 and 2 give the rate of execution s_{inf} for the four different memory latencies for TPCC and SPECJBB respectively in cycles per reference. We conclude that the variations of s_{inf} are negligible.

Figures 1 and 3 depict cumulative distribution functions (cdfs) of the α and τ distributions for the three reference types for TPCC and SPECJBB respectively. Note the logarithmic horizontal scale of the α distribution graphs. On each plate cdfs corresponding to the four different memory latencies are overlaid. In the case of τ_{st} , the graphs do not reach the ordinate of one because a nonnegligible percentage of stores does not cause a stall. It is easily seen that latency changes cause indistinguishable differences to five TPCC distributions and all SPECJBB distributions. TPCC τ_{st} in Figure 1f shows light sensitivity to latency for large distances while still being invariant for small distances. We conjecture that this is because we fail to observe the store behavior for large miss-to-stall times for small system latencies. When we multiply the upper bound of the interval of agreement among the four curves in Figure 1f (about $x_\alpha = 17$) by the rate of execution, $\frac{x_\alpha \times s_{\text{inf}}}{2.8\text{GHz}} = \frac{17 \text{ refs} \times 7.20 \text{ cycles/ref}}{2.8 \text{ cycles/ns}} = 44$ ns, we get close to the smallest system latency considered (58 ns). Therefore, we reach an agreement in the common region of observation. A simple remedy for this problem is to use τ_{st} that corresponds to a large latency (e.g. 247 ns) in the abstraction. We do not find this problem in the case of SPECJBB where a very small percentage of stores causes stalls after 58 ns (see Figure 3f).

Table 2: SPECJBB: Relative errors of the miss frequencies for the three reference types. a. Abstractions are for the particular latency in each case. b. The same abstraction is used throughout.

latency	load	i-fetch	store	s_{inf}	latency	load	i-fetch	store
1	1.01	1.02	1.00	4.57	1	1.03	1.00	1.03
11	1.01	1.02	1.01	4.57	11	1.02	1.00	1.03
30	1.01	.99	1.01	4.59	30	1.02	1.00	1.02
190	.98	.99	.99	4.67				

(a)
(b)

Next, we study the accuracy of the proposed model. Here we compare the model results against those given by cycle-accurate simulation. Let $r_{ld,l}^{\text{mod},l'}$ be the number of L2 load misses issued per second in a system with memory latency l' as computed by the model when using the abstraction that corresponds to memory latency l . That is, the abstraction is computed using a trace from simulating a system with latency l . We then use this abstraction to model a system with a possibly different latency l' . Furthermore, let $r_{ld,l}^{\text{sim}}$ be the corresponding quantity given by the cycle-accurate simulation of a system with latency l . First, we use the same latency l for both the abstraction and the modeled system. The load column in the left half of Table 1 presents TPCC ratios $r_{ld,l}^{\text{mod},l}/r_{ld,l}^{\text{sim}}$ for the four latencies under consideration. The instruction and store column entries are defined analogously. The left half of Table 2 contains the corresponding numbers for SPECJBB. We observe that the errors range from 0% to 3%. This implies that the abstraction captures almost all important information about the processor as it impacts system performance. Second, we investigate the effect of using a fixed abstraction to model systems with different latencies. This ability is crucial because our goal is to model a variety of interconnects with a single processor abstraction. Since latency invariance of the abstraction primitives has already been shown, we do not expect any notable changes in the model accuracy. The right halves of Tables 1 and 2 present ratios $r_{ld,247}^{\text{mod},l}/r_{ld,l}^{\text{sim}}$ for the two workloads respectively. That is, we use the same abstraction obtained for the average memory latency $l = 247$ ns (corresponding to setting the DRAM latency to 190 ns) to model systems with the other three latencies. Indeed, we observe similarly small errors.

Finally, we vary the size of the L2 cache between 256KB and 2MB to show that the abstraction is insensitive to changes in the cache configuration. Figure 4 depicts the α abstraction distributions for both workloads and the four cache sizes of 256KB, 512KB, 1MB, and 2MB. The distributions are exponentially rescaled as described in Section 2 for modeling the 256KB cache. Results for modeling the other caches are qualitatively very similar. Despite slight variations, the errors incurred when using a unique abstraction to model systems with different cache sizes fall into the range of Tables 1 and 2 (up to 4%) and are not shown. Also recall that these distributions can be obtained via cache simulation for each cache configuration separately. The τ distributions show agreement and are not depicted.

4. Conclusion

In this work we proposed a method of modeling a processor in a generic high level system model. We extend previous work in several ways. First, the component model is portable

for use in a variety of system modeling contexts rather than being hardwired into a specific modeling methodology and can be developed and extended in isolation.

Second, it is based on abstracting the processor activity that is relevant for performance modeling. High level simulation using this model probabilistically generates a typical execution run including cache miss overlap of different type misses, processor stalls, and miss burstiness. Here we need not make approximations or questionable assumptions of invariances that are typical in the literature (e.g. the epoch model discussed in the introduction). Instead we find the invariances that are inherent to system behavior. They are intuitive and present a compact description of the interaction of the processor and the memory subsystem. At the same time they provide extremely accurate estimates of system performance.

Third, this methodology permits straightforward extensions to account for advanced architectural features. For example, we modeled instruction prefetching in Opteron within the same framework. In addition, the model permits easy and clean assessment of benefits of these features. It provides insights into the way they improve performance by generating a process that is typical of the new execution pattern. In the instruction prefetch example, it would be straightforward to determine stochastically how many other misses can be overlapped with a prefetched instruction miss, which would otherwise stall the processor immediately.

As discussed at length above, the same abstraction is suitable for modeling any system interconnect including multiprocessors, configurations with many levels of cache hierarchy, and various coherence protocols and constructs. By contrast, the abstraction primitives are obtained by parsing through a single trace obtained from single core simulation.

In this work we did not consider other advanced processor features, in particular, runahead execution. Traditionally, runahead execution analysis has been carried out through detailed processor modeling [2, 10]. We believe that high level models can be developed within the proposed framework. This amounts to characterization of stall period activity in the ways similar to the proposed one. This would be an interesting item for future research. Another future research item mentioned above is the development of extrapolated interarrival miss distributions α in addition to miss rate extrapolation as in Gluhovsky and O’Krafka [5].

Acknowledgments

The author is indebted to Jimmy Williams for supplying the data used to validate the model and for his many helpful suggestions. The author would also like to thank Brian O’Krafka, Ben Fuller, and Pat Conway for many helpful comments and John Busch for his support and encouragement.

References

- [1] Barre, J., Landet, C., Rochange, C., and Sainrat, P. , Modeling Instruction-Level Parallelism for WCET Evaluation. In *Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing, Systems and Applications*, 61-67, 2006.
- [2] Chou, Y., Fahs, B., and Abraham, S., Microarchitecture Optimization for Exploiting Memory-Level Parallelism. In *Proceedings of the 31th International Symposium on Computer Architecture*, 2004.
- [3] Chou, Y., Spracklen, L., and Abraham, S., Store Memory-Level Parallelism Optimizations for Commercial Applications. In *Proceedings of the IEEE/ACM Int. Symp. Microarchitecture.*, 2005.

- [4] Eager, D.L., Sorin, D.J., and Vernon, M.K., AMVA Techniques for High Service Time Variability. In *Proceedings of 2000 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, 217-228, 2000.
- [5] Gluhovsky, I., and O’Krafka, B.W., Comprehensive Multiprocessor Cache Miss Rate Generation Using Multivariate Models. *ACM Transactions on Computer Systems* 23(2), 111-145, 2005.
- [6] Kaplan, E.L. and Meier, P., Non-Parametric Estimation from Incomplete Observations. *Journal of the American Statistical Association*, 53, 457-481, 1958
- [7] Karkhannis, T.S. and Smith, J.E., A First-Order Superscalar Processor Model. In *Proceedings of the 31th International Symposium on Computer Architecture*, 2004.
- [8] Keltcher, C.N., McGrath, K.J., Ahmed, A., and Conway, P. The AMD Opteron Processor for Multiprocessor Servers. *IEEE Micro*, 66-76, 2003.
- [9] Kunkel, S., Eickmeyer, R., Lipasti, M., Mullins, T., O’Krafka, B., Rosenberg, H., Vanderweil, S., Vitale, P., and Whitley, L. A Performance Methodology for Commercial Servers. *IBM Journal of Research and Development* 44(6), 851-873, 2000.
- [10] Mutlu, O., Kim H., Armstrong, D. N., Patt, Y. N. An Analysis of the Performance Impact of Wrong-Path Memory References on Out-of-Order and Runahead Execution Processors, *IEEE Transactions on Computers*, 54(12), 1556-1571, 2005.
- [11] Sharapov, I., Kroeger, R., Dalamarter, G., Cheveresan, R., and Ramsay, M., In *Proceedings of the 11th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 81-89, 2006.
- [12] Sorin, D., Pai, V., Adve, S., Vernon, M., and Wood, D., Analytic Evaluation of Shared-Memory Systems with ILP Processors. In *Proceedings of the 25th International Symposium on Computer Architecture*, 380-391, 1998.
- [13] Sorin, D.J., Lemon, J.L., Eager, D.L., and Vernon, M.K., A Analytic Evaluation of Shared-Memory Architectures. *IEEE Transactions on Parallel and Distributed Systems* 14(2), 166-180, 2003.
- [14] Tsuei, T.-F., Yamamoto, W., Queuing Simulation Model for Multiprocessor Systems. *IEEE Computer* 36(2), 58-64., 2003.

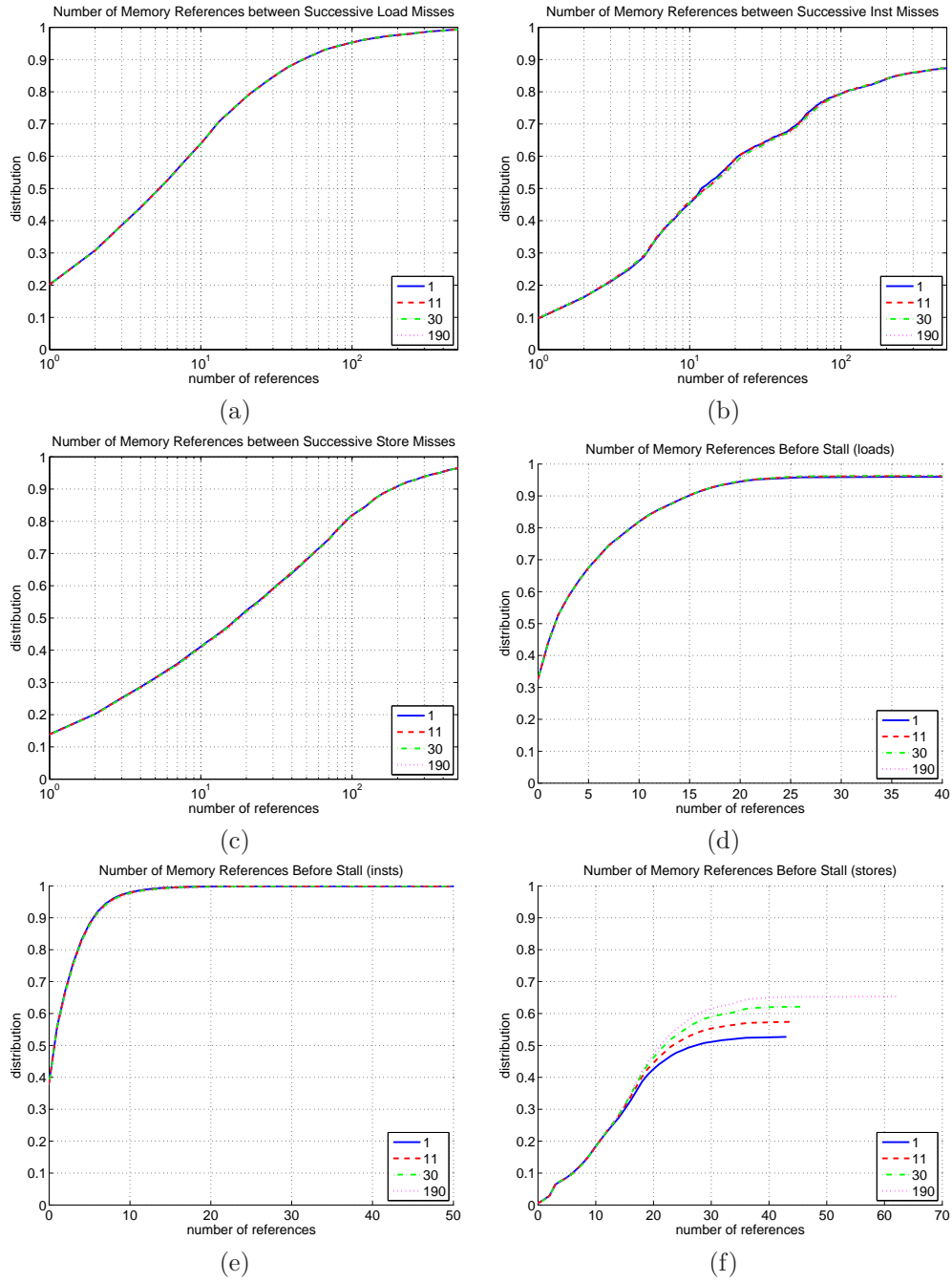


Fig. 3: SPECJBB abstraction distributions for 4 levels of memory latency. First row: a. α_{ld} , b. α_{if} ; Second row: c. α_{st} , d. τ_{ld} ; Third row: e. τ_{if} , f. τ_{st}

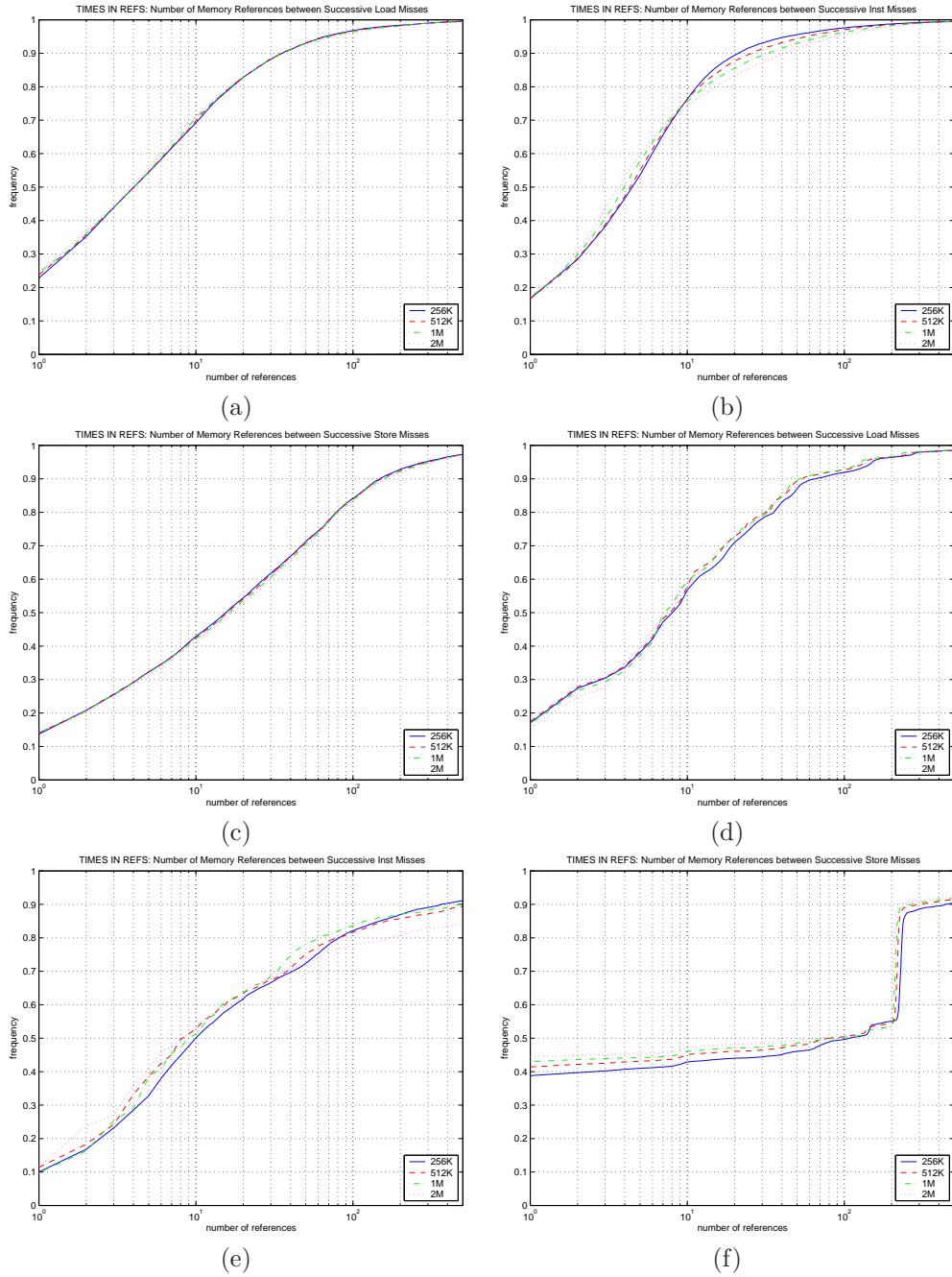


Fig. 4: Rescaled abstraction distributions α for four different cache sizes for both TPCC and SPECJBB and the DRAM latency of 30 ns. First row: a. TPCC α_{ld} , b. TPCC α_{if} ; Second row: c. TPCC α_{st} , d. SPECJBB α_{ld} ; Third row: e. SPECJBB α_{if} , f. SPECJBB α_{st} ,